Polecenie: Napisz algorytm obliczający silnie metodą iteracyjną oraz rekurencyjną.

Na początku wczytałem podstawową bibliotekę <stdio.h>, oraz zdeklarowałem 2 funkcje:

```
double silnia_iteracyjnie(int n)
{
long long silniaIteracyjna = 1;
for (n; n > 0; n--)
{
silniaIteracyjna = silniaIteracyjna * n;
}
return silniaIteracyjna;
}
```

oraz

```
double silnia_rekurencyjnie(int n)
{
if (n < 2)
return 1;

return n * silnia_rekurencyjnie(n - 1);
}
```

Zacznijmy od funkcji iteracyjnej. Na początku w „ciele” funkcji określiłem zmienną typu long ling o nazwie siniaIteracyjna i ustawiłem jej wartość na 1. Następnie zdefiniowałem pętle typu for której wartość n jest równa liczbie z której ma zostać policzona silnia.

Zmienną n z każdym przejściem pętli zmniejszałem o 1 aż do momentu gdy n>0. W ciele pętli do zmiennej silniaIteracyjna wpisywałem jej własną wartość * n. Na końcu funkcja zwraca wartość silniaIteracyjna. Dodatkowo w całej funkcji jest dodatkowo pętla która sztucznie wydłuża czas wykonania funkcji. Dzięki niej możliwe jest sprawdzenie czasu wykonania programu.

Teraz czas na funkcję rekurencyjną. Na początku określiłem warunek że n<2, jeśli to prawda to funkcja zwraca wartość 1. Jeśli jest to fałsz to silnia zwraca wartość: n*silnia_rekurencyjnie(n-1) czyli odwołuje się do samej siebie tylko zmniejsza n (n-1).

Dodatkowo w całej funkcji jest dodatkowo pętla która sztucznie wydłuża czas wykonania funkcji. Dzięki niej możliwe jest sprawdzenie czasu wykonania programu.

Następnie wprowadziłem funkcję main() w której określiłem `int narg, char *argv[]`
będą one potrzebne do wyboru metody poleczenia silni int main(int narg, char *argv[])
Później określiłem zmienne pomocnicze oraz zmienną double silnia która zwróci wartość policzonej silni.
W dalszej części określiłem instrukcję warunkową switch dzięki której użytkownik może wybrać czy korzysta z metody 1 czy 2 (musi wpisać -M1 albo -M2).

Następnie program pobiera dane w postaci -Mx gdzie x to 1 albo 2 i jest to określenie metody oraz sama liczba z jakiej ma zostać policzona silnia.

Na koniec program wywołuje odpowiednią funkcję gdzie metoda 1 to silnia rekurencyjna, natomiast metoda 2 to silnia iteracyjna i wypisuje wynik.

Wynik jest zwracany jako double dzięki czemu możliwe jest licznie silni z liczby ponad 21!, jeśli byłaby to zmienna np. int wtedy doszłoby do przepełnienia. Dzięki double program liczy silnie nawet z 60!.

==========

Przykłady działania programu:

```
bartosz@Baxing-VB:~/Desktop/C/Zestaw2$ ./zad1.out -M2 3
Wynik z 3! = 6 (użyta metoda: 2)
```

czyli dla M1 silnia jest liczona przez rekurencję, a więc mamy
3*(3-1)*((3-1)-1) a więc wynikiem będzie 6.

---

przykład działania programu iteracyjnego:

```
bartosz@Baxing-VB:~/Desktop/C/Zestaw2$ ./zad1.out -M1 13
Wynik z 13! = 6227020800 (użyta metoda: 1)
```

mamy zmienną x=1;
następnie mamy:
x=1*1 (1)
x=1*2 (2)
x=2*3 (6)

POZOSTAŁE PRZYKŁADY DLA OGROMNYCH LICZB:

```
bartosz@Baxing-VB:~/Desktop/C/Zestaw2$ ./zad1 -M1 50
Wynik z 50! = 30414093201713375576366966406747986832057064836514787179557289984
 (użyta metoda: 1)
```

```
bartosz@Baxing-VB:~/Desktop/C/Zestaw2$ ./zad1 -M1 60
Wynik z 60! = 832098711274139158005639610295964107745794554107670881359908535053
31187384917164032 (użyta metoda: 1)
```

PORÓWNANIE CZASÓW WYKONANIA: **widać że -M2 (iteracja) jest dużo szybsza**

```
bartosz@Baxing-VB:~/Desktop/C/Zestaw2$ time ./zad1 -M1 100000
Wynik z 100000! = inf (użyta metoda: 1)

real    1m9,253s
user    1m9,218s
sys     0m0,000s
bartosz@Baxing-VB:~/Desktop/C/Zestaw2$ time ./zad1 -M2 100000
Wynik z 100000! = 0 (użyta metoda: 2)

real    0m17,463s
user    0m17,448s
sys     0m0,008s
```